Cache

Haitao Wei

(All these slides were from

Daniel Orozco)

University of Delaware
http://www.udel.edu

Computer Architecture and
Parallel Systems
Laboratory
http://www.capsl.udel.edu

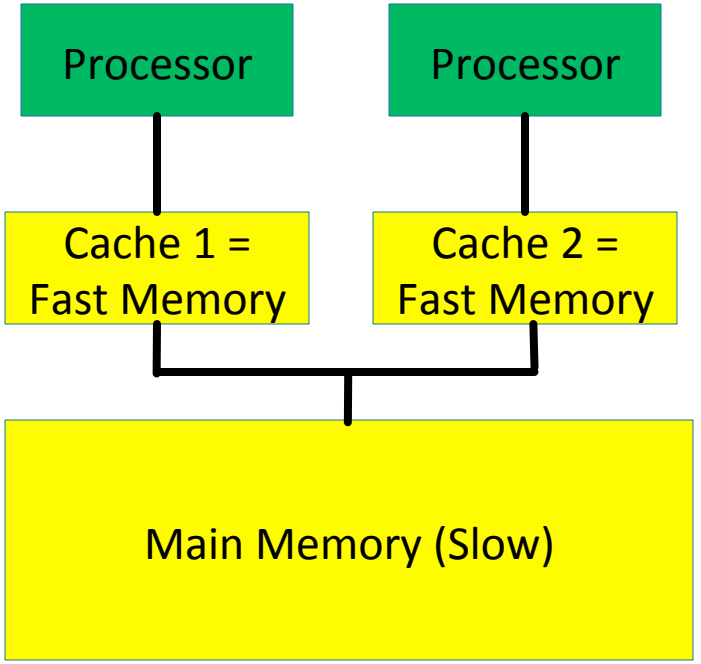
Cache: A short review

Cache: A small, fast memory that holds values that are frequently used.

Data Locality: A property of a program: Memory accesses are close.

Temporal Locality: Close in time.

Spatial Locality: Close in address.

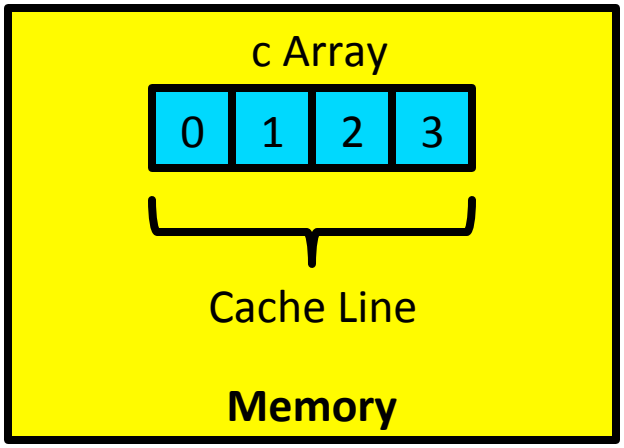


Cache Line

The idea of Cache: Put things in the fast local memory to make them run fast.

To simplify the design of the machine, the minimum unit of transfer to a cache is a **cache line.**

The processor can read or write less than one line.

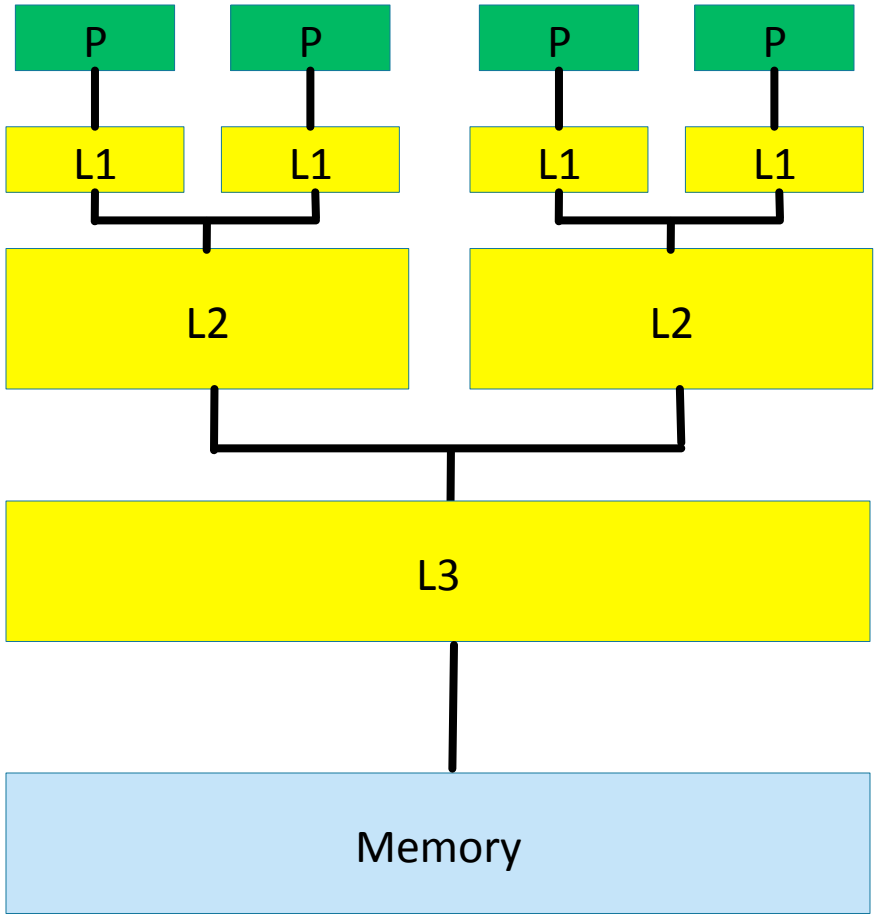


Writing to c[1] has the same memory effect as writing to the whole line.

Cache Hierachy

Faster Memory: More expensive, more power, harder to make.

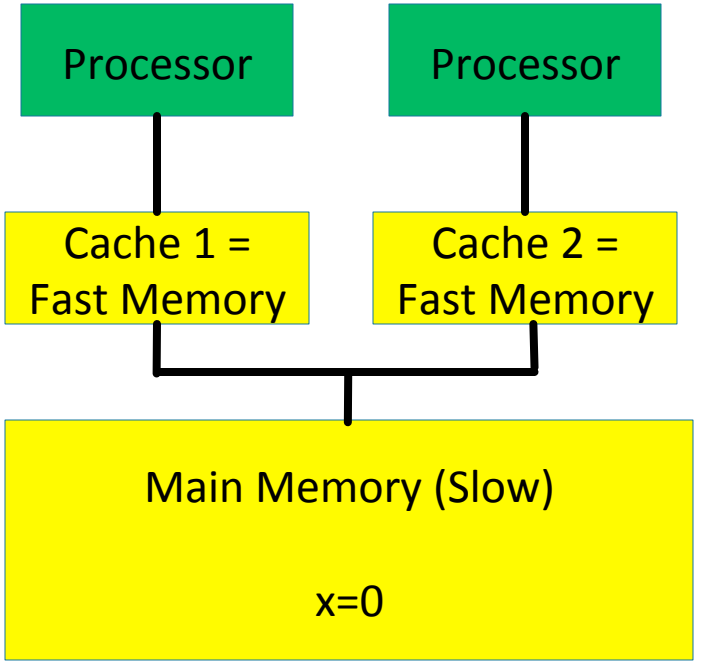
Architects use multiple levels of cache to go from very fast (L1 cache) to slow (L3 cache and memory).



Cache: A short review

Processors that do not share cache suffer from many **cache conflicts** when trying to access the same variable.

Cache Invalidation: You have a copy of the variable in your cache and another processor writes to it.

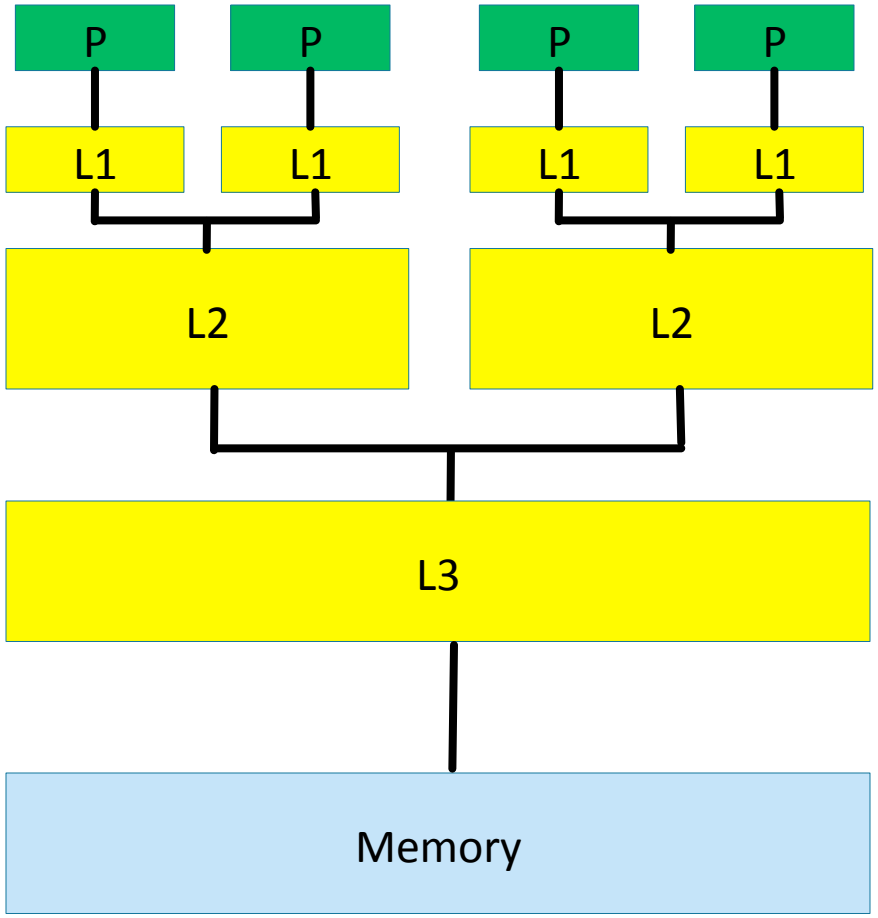


CACHE PROPERTIES

Cache Hierarchy

Faster Memory: More expensive, more power, harder to make.

Architects use multiple levels of cache to go from very fast (L1 cache) to slow (L3 cache and memory).



Cache Associativity

What line does x and y go?

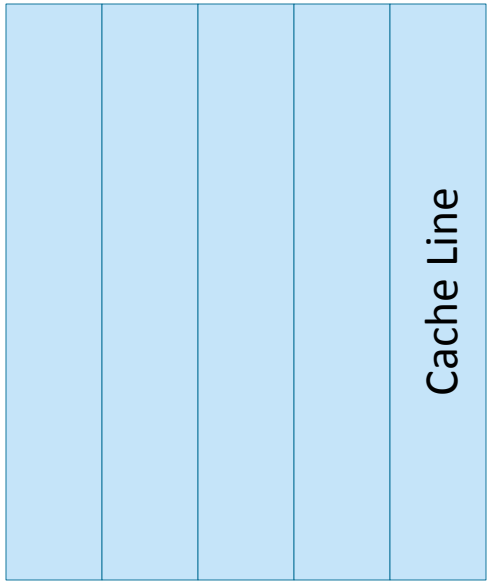
This is the cache Associativity concept: How many different lines can receive a particular value.

Usually, Cache Associativity = 4 or 8.

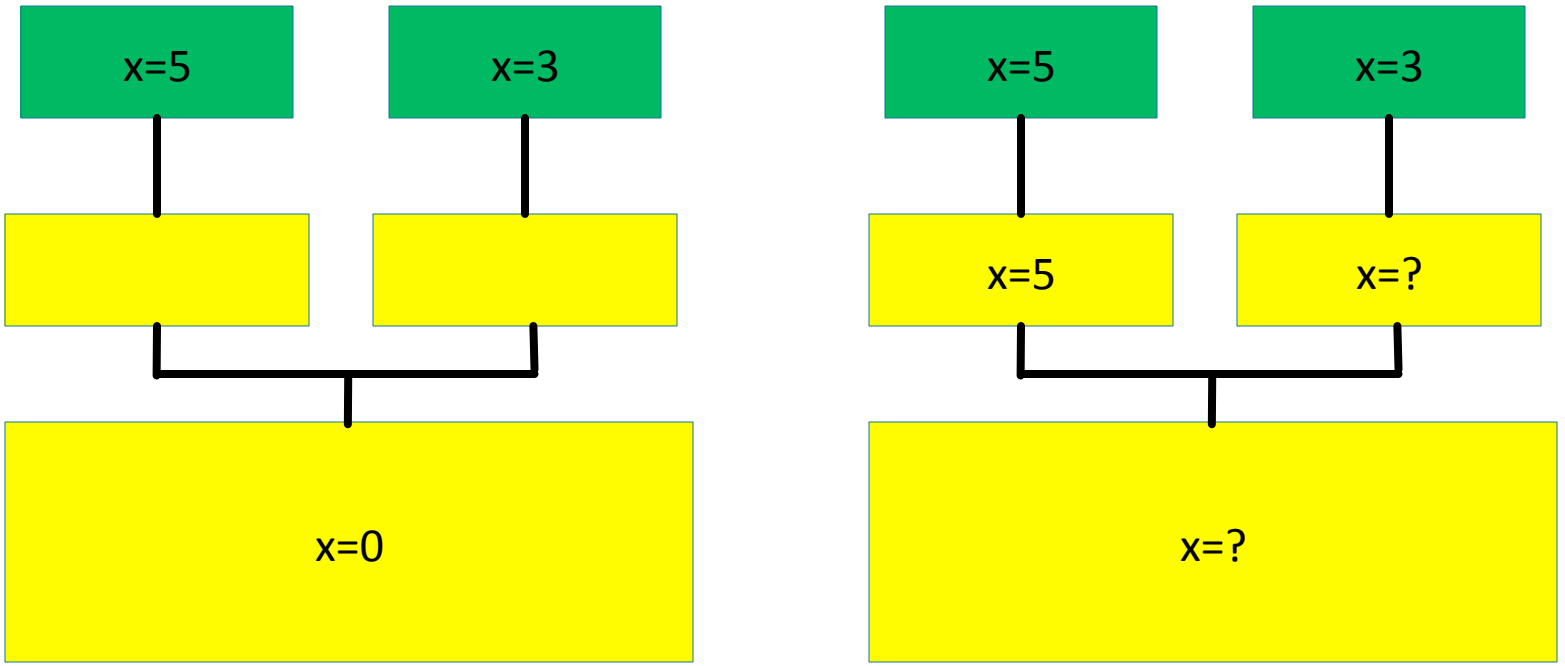
When Cache Associativity = infinite we say **"A fully associative cache"**

x: Address 0x3956

y: Address 0x4956



Cache Coherence



The property of having a correct value in the cache is called “Cache Coherency”. The definition of **correct** is given by the **memory model. Cache Coherency is one of the big challenges** of computer architecture.

CACHE COHERENCE PROTOCOLS

The Cache Coherence Definition

The memory model should specify what happens when multiple processors execute several writes and reads.

The Sequential Consistency rules imply:

A processor writing and reading from his cache observes its effect.

A read or a write to a memory location should see the last write to that memory location.

Cache Coherence

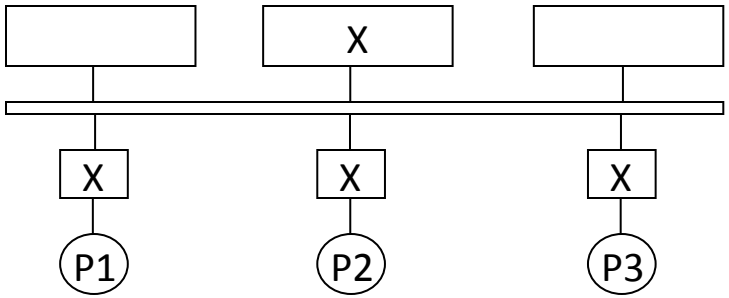
The Coherence Problem (under Sequential Consistency)

- A processor should have exclusive access to a shared variable when writing and should get the "most recent" value when reading.

Solution

- When writing
  - (1) **Invalidate** all copies
  - (2) **Broadcast** to everyone the new copy
- When reading
  - Find the most recent copy
    - Can be tricky

Write Update V.S Write Invalidate



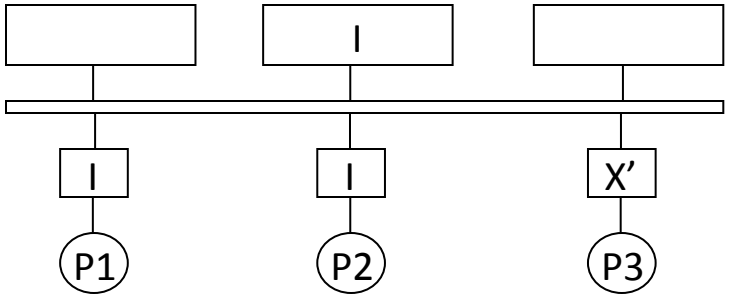
Shared Memory

Bus

Cache

Processors

X is a shared variable that has a copy in all caches. Then a write occurred



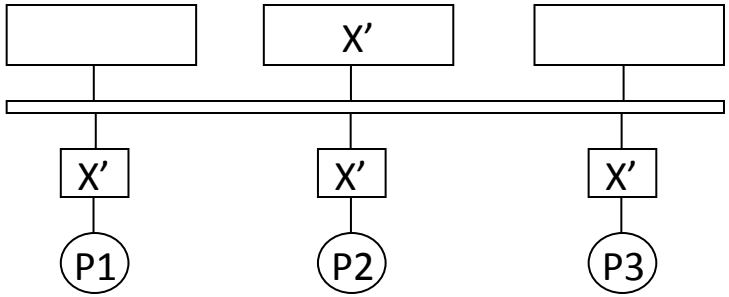
Shared Memory

Bus

Cache

Processors

For **Write Invalidate** all the cache copies are marked as "invalid" except the most recent one



Shared Memory

Bus

Cache

Processors

For **Write Update** all the cache copies are updated with the most recent value

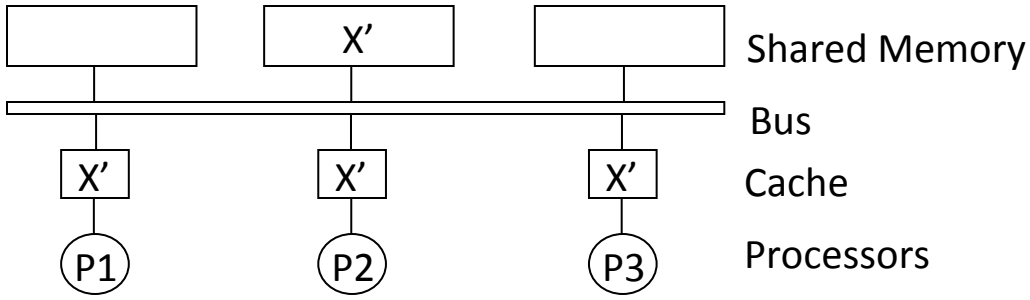
Assume a write through cache protocol

Coherence Protocols: Snooping

All caches can "Observe" the memory operations of all processors to:

- Keep values updated.
- Know the values are invalid

Higher levels of cache can rely on lower levels of cache to get correct values.

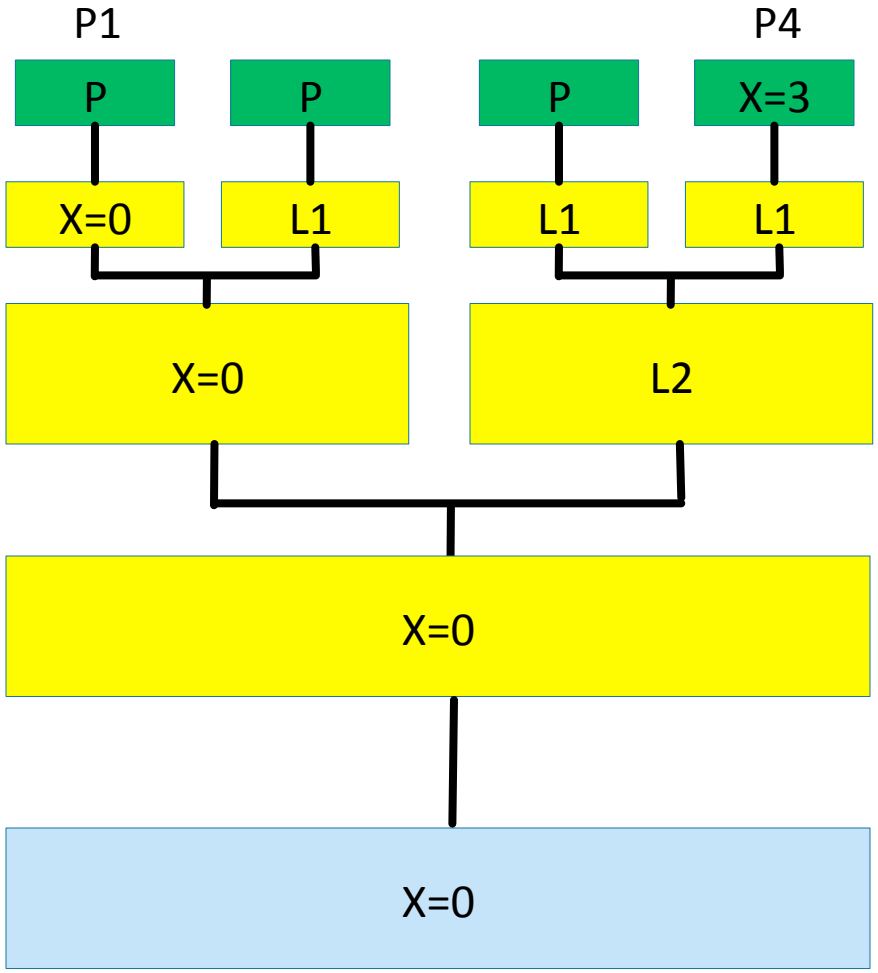


Snooping Example

When P4 writes X=3:

- L1 -> L2 -> L3 request for X
- Write to L1 (X=3)
- L1 notifies L2: Invalid
  - May also update L2:X=3
- L2 notifies L3: Invalid
  - May also update L3:X=3
- May also update memory. M:X=3

P1 can get the value from the L2 bus.



Snooping is not very scalable

The problem with snooping is that it requires a **bus.**

Bus: A shared medium:

- Only one processor can use it in one cycle
- Tricks can be used
- No tricks for 10000 processors ☹

In general, not practical for more than 32 processors.

Snooping Protocol

Write Invalidate

- Writing Processor sends an invalidation signal
- Caches that are listening to the bus will invalidate their copy of such variable
- The writing processor writes to the variable
  - Memory is updated according to which cache protocol policy you have
    - Write through, Write Back

Write Update

- Writing processor will broadcast the new value to all caches

Directory Based Protocols

Reason

- Bus based protocols may generate too much traffic
- Multi level Interconnection Network may not have efficient broadcasting capabilities as a system bus

Directory Based system

- A directory with an entry per memory location
- Central vs Distributed

Cache Protocols: Directory Based

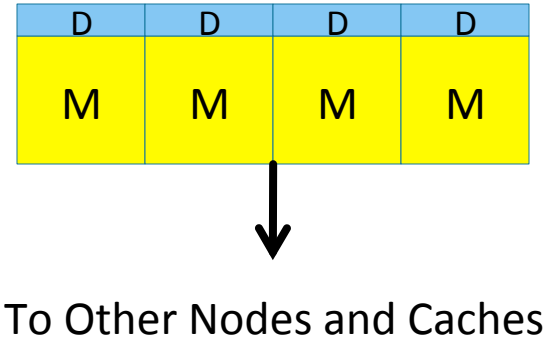
Who has cached X?

A directory is kept!

Typically, the directory is kept in main memory next to the block of memory.

A directory is usually simple: An array with information of ownership from each processor.

This is a parallel implementation that scales with the size of the system.



Directory Based Protocols

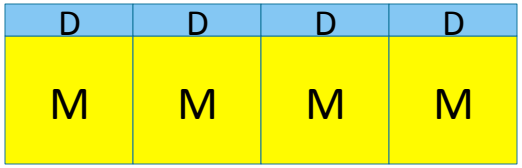
Information about ownership:

- A bit indicating user,
- A bit indicating state

Messages are interchanged between processors to control ownership.

Messages can be, for example:

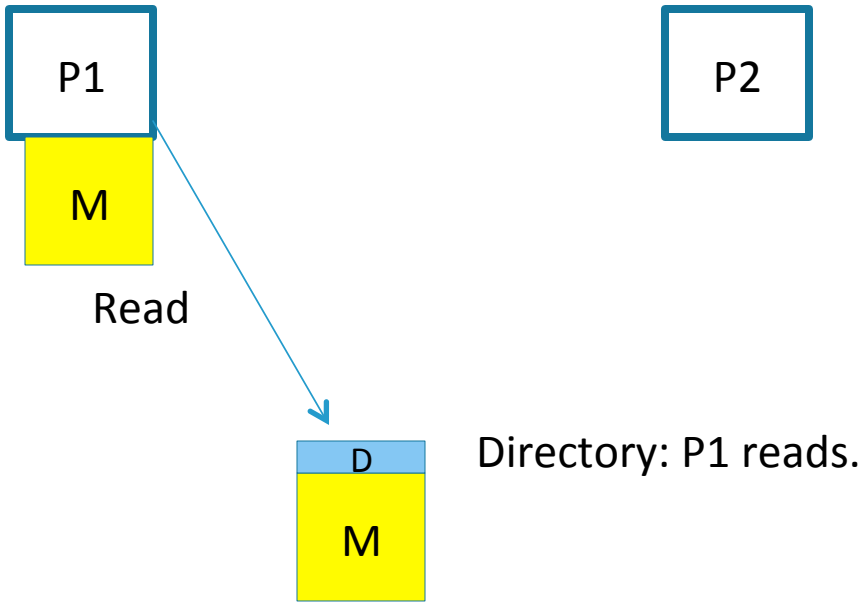
- Request for block
- Acquire
- Invalidate



Ensuring Coherence May be Complicated.

What happens when two processors try to write to the same memory block?

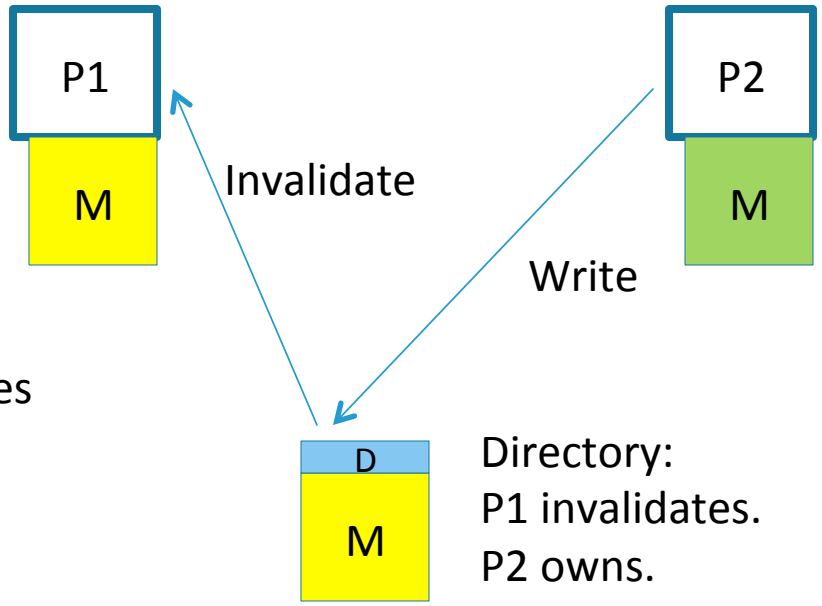
Example:



Ensuring Coherency May be Complicated.

What happens when two processors try to write to the same memory block?

Example:



What happens if P1 writes (or reads again) the memory during the time the message travels the network?

Directory:
P1 invalidates.
P2 owns.

Cache Coherence Protocol

Directory Based:

- The info about one block of physical memory is kept in a single location
- The directory itself can be distributed
- Advantages
  - Scalable
- Proportional to Main Memory Size

Bus Based Snooping

- Use the shared memory bus
- Every cache that has a copy of the data is responsible to maintain coherence about it
- Advantages
  - Easily add on to existent busses
- Proportional to cache size

AN INTRO TO THE "MSI" FAMILY

Requirement for Implementation

Stores to each memory location occurs in program order

All processing elements see such order if they access the same memory location

Only one processor has the write privilege at a time.

Implementation

If Read(X)

- Hit: Just copy
- Miss: Find the location residence of X
  - Memory, other cache, others
  - Receive a legal copy

If Write(X)

- Acquire Ownership and / or Exclusivity

Note: Assume a Write Back – Write Invalidate Cache Protocol

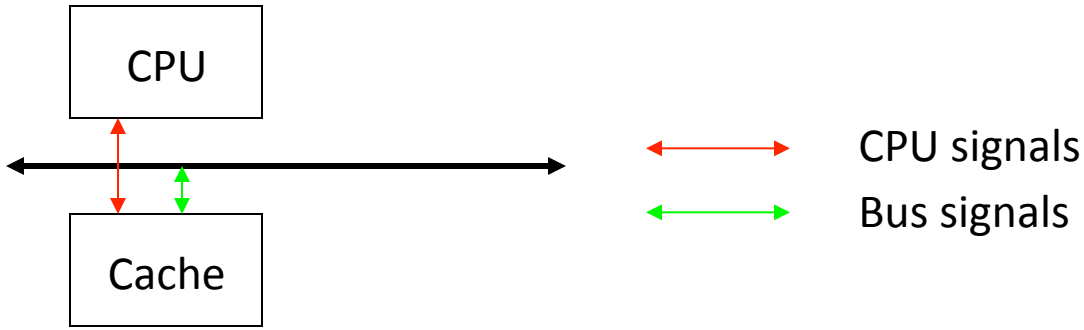
Finite Automata for Cache Protocols

State Transactions

- Read Misses
- Write Hits
- Write Misses

Type of Outputs

- Bus Signals and CPU actions



Cache Protocols as Finite Automata

Finite Automata

- A graph in which vertex are states and edges are transitions
- Usually, a transition (an edge) will be labeled with a 2-tuple a / b where a is the input action that produced the state change and b is an action that will result from this state change.
  - The input action may be, in fact, many actions, the same goes to the output action

Finite Automata as cache protocols

- A vertex is the state of a given cache line
- The transitions may be produced by
  - Processor actions
  - Bus signals

The MSI Protocol

Similar to the protocol used by Sillicon Graphics 4D series of multiprocessors machines

Three states to differentiate between clean or dirty

- Modified, Shared and Invalid

Two types of Processor actions and Three types of bus's signals

- Processor Writes and Reads
- Bus Read, Bus Read Exclusive and Bus Write Back

MSI States

Modified

- The cached copy is the only valid copy in the system.
- Memory is stale.

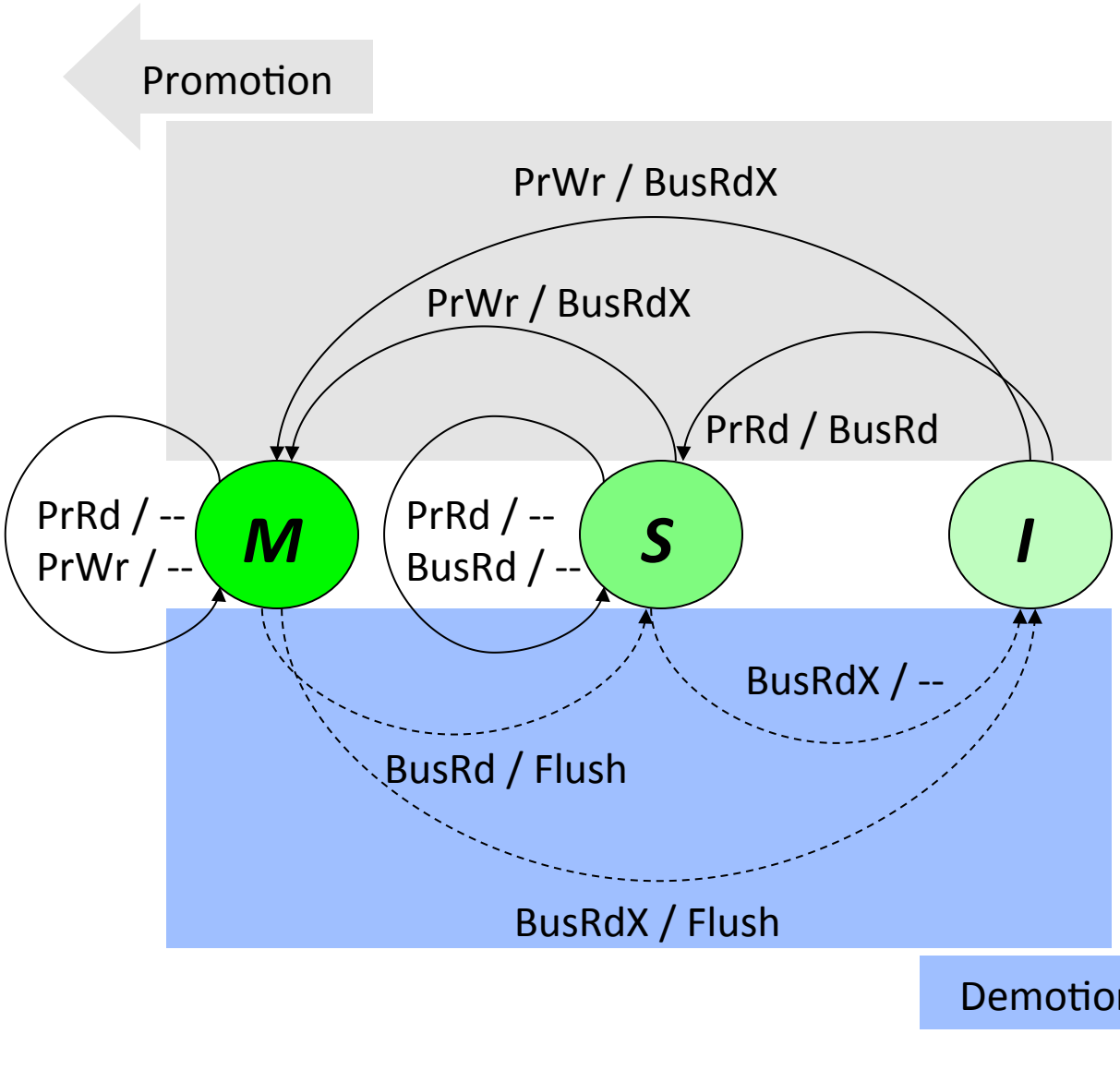
Shared

- The cached copy is valid and it may or may not be shared by other caches.
  - Initial state after first loaded.
- Memory is up to date.

Invalid

- The cached copy is not valid: An invalidation signal was received..

MSI Protocol State Machine



Input / Output

PrWr	Processor Write
PrRd	Processor Read
BusRd	Bus Read
BusRdX	Read to own
Flush	Flush to memory
--	No Action

MSI Example

Process or Action	State P1	State P2	State P3	Bus Action	Data Supplied by
P1 loads u	S	_	_	BusRd	Mem
P3 loads u	S	_	S	BusRd	Mem
P3 stores u	I	_	M	BusRdX	Mem
P1 loads u	S	_	S	BusRd	P3 cache
P2 loads u	S	S	S	BusRd	Mem

The MESI Protocol

States:

- Modified, Exclusive, Shared and Invalid

Due to Goodman [ISCA'93]

State transitions are due to:

- Processor actions: This being Write or Reads
- Bus operations caused by the former

Implemented in Intel Pentium Pro (in some modes)

MESI States

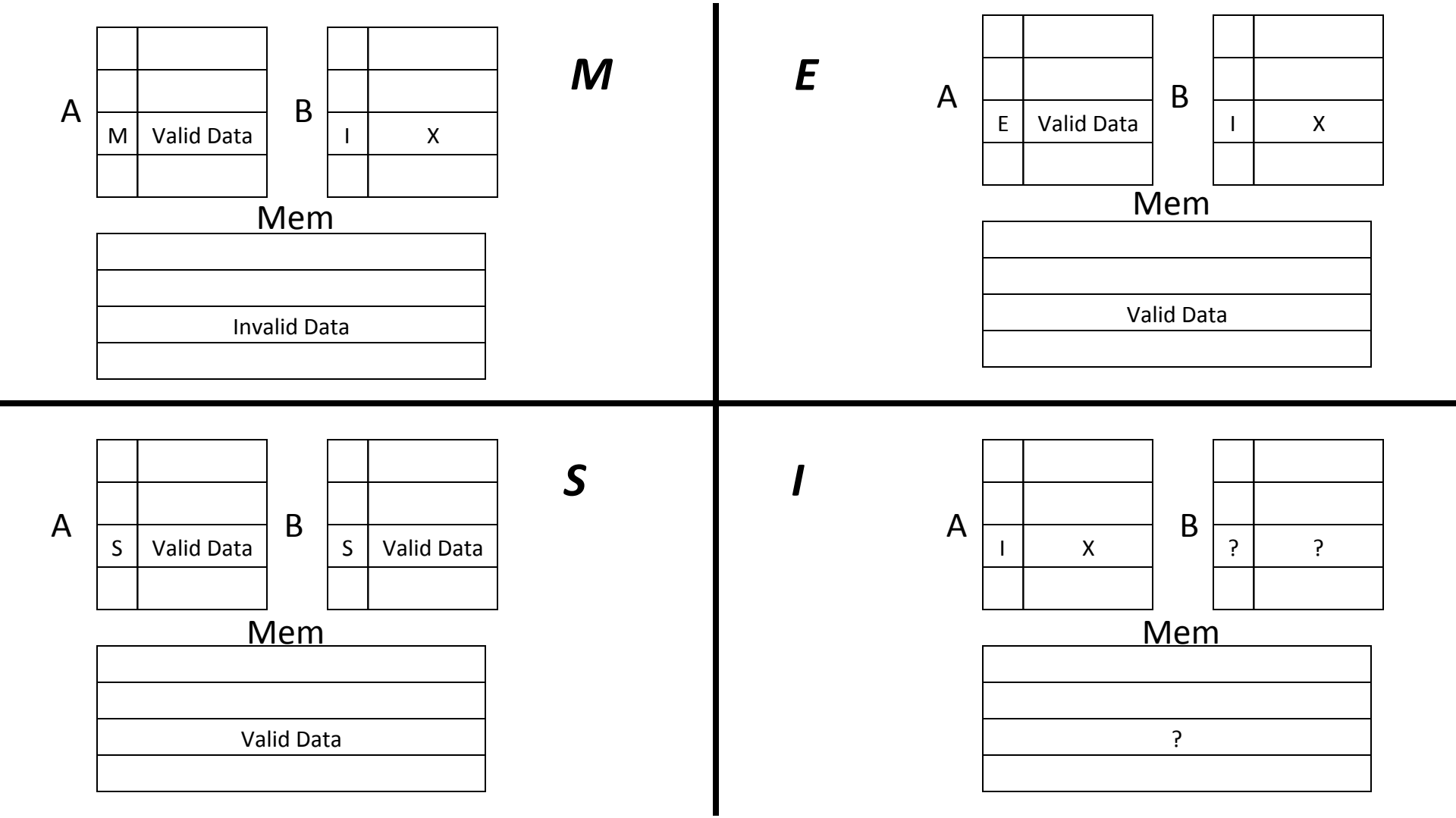
Modified
- Main Memory's value is stale
- No other cache possesses a copy

Exclusive
- Main Memory's value is up to date
- No other cache possesses a copy

Shared
- Main Memory's value is up to date
- Other caches have a copy of the variable

Invalid
- This cache have a stale copy of the variable

MESI States


M
A
M
Valid Data
B
I
X
Mem
Invalid Data
E
A
E
Valid Data
B
I
X
Mem
Valid Data
S
A
S
Valid Data
B
S
Valid Data
Mem
Valid Data
I
A
I
X
B
?
?
Mem
?

Example: Two Processor System

Cache 1		Cache 2		Memory Transfer
Bus	State	Bus	State	
	I		I	Load into Cache 1
	I → E		I	

P1 ➔ Load

Cache 1		Cache 2		Memory Transfer
Bus	State	Bus	State	
	E		I	Load into Cache 2
Rd Hit	E → S		I → S	

P2 ➔ Load

Cache 1		Cache 2		Memory Transfer
Bus	State	Bus	State	
	S		S	
	S →M	Inv	I	

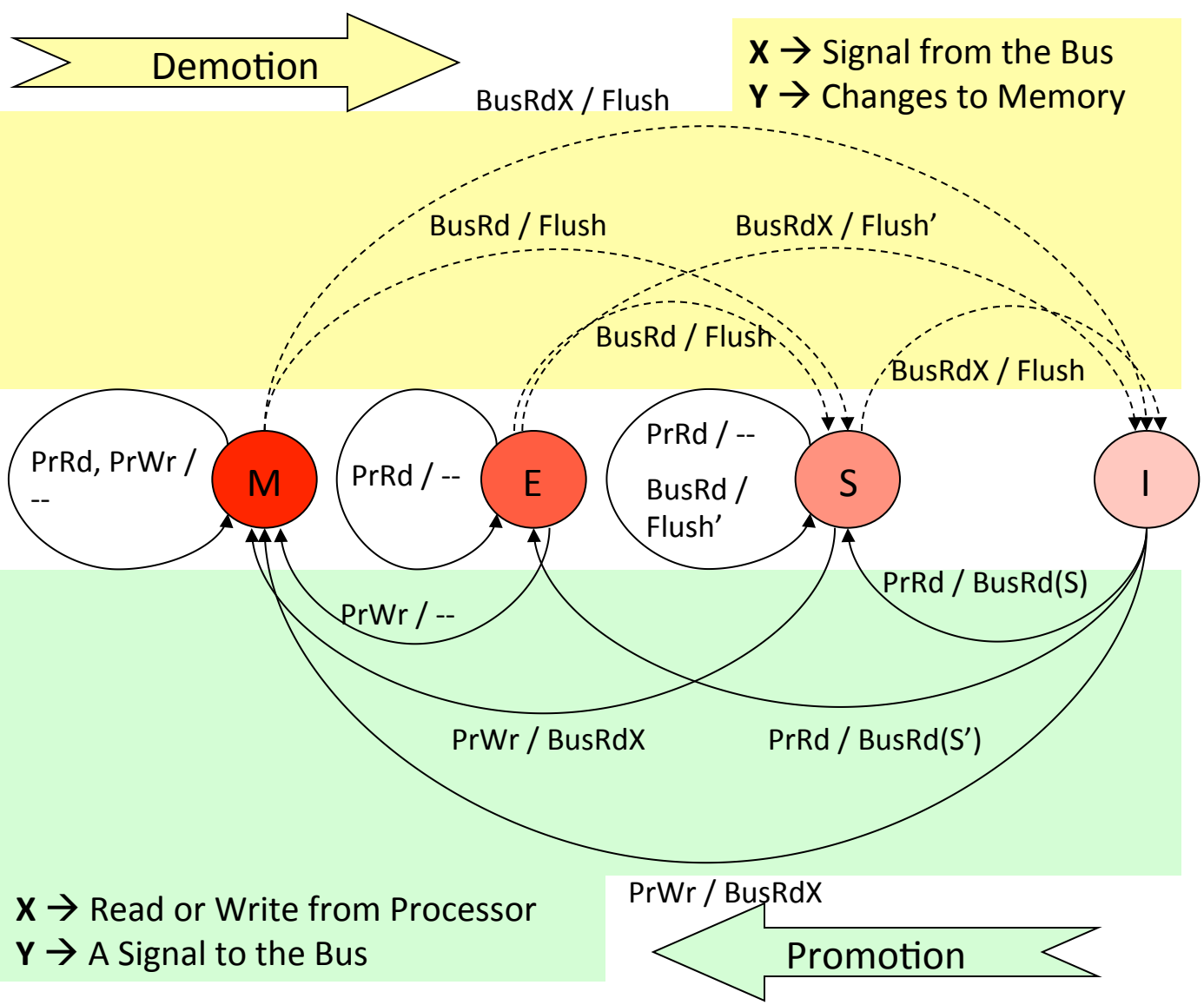
P1 ➔ Store

Cache 1		Cache 2		Memory Transfer
Bus	State	Bus	State	
	M		I	Store from Cache 1 Load into Cache 2
	M		I	
Rd Hit	M → S		I	
	S		I → S	

P2 ➔ Load (first abort and then try again)

MESI Protocol State Machine

*Extracted from "**Parallel Computer Architecture: A Hardware & Software Approach**" by Culler & Singh. Page 301*



X / Y

X is the input
Y is the Output

Inputs and Outputs

PrRd A Processor Read

PrWr A Processor Write

BusRdX

A Bus Read Exclusive. Request the data to be exclusive to this cache or demote it to a shared state

BusRd(S)

A Bus Read when the element is shared by another processor

BusRd(S')

A Bus Read when the element is not shared by another processor

Flush

Flush to either memory or a requesting processor (according to what sharing scheme is used)

Flush'

Flush to either memory or a requesting processor (only that processor)

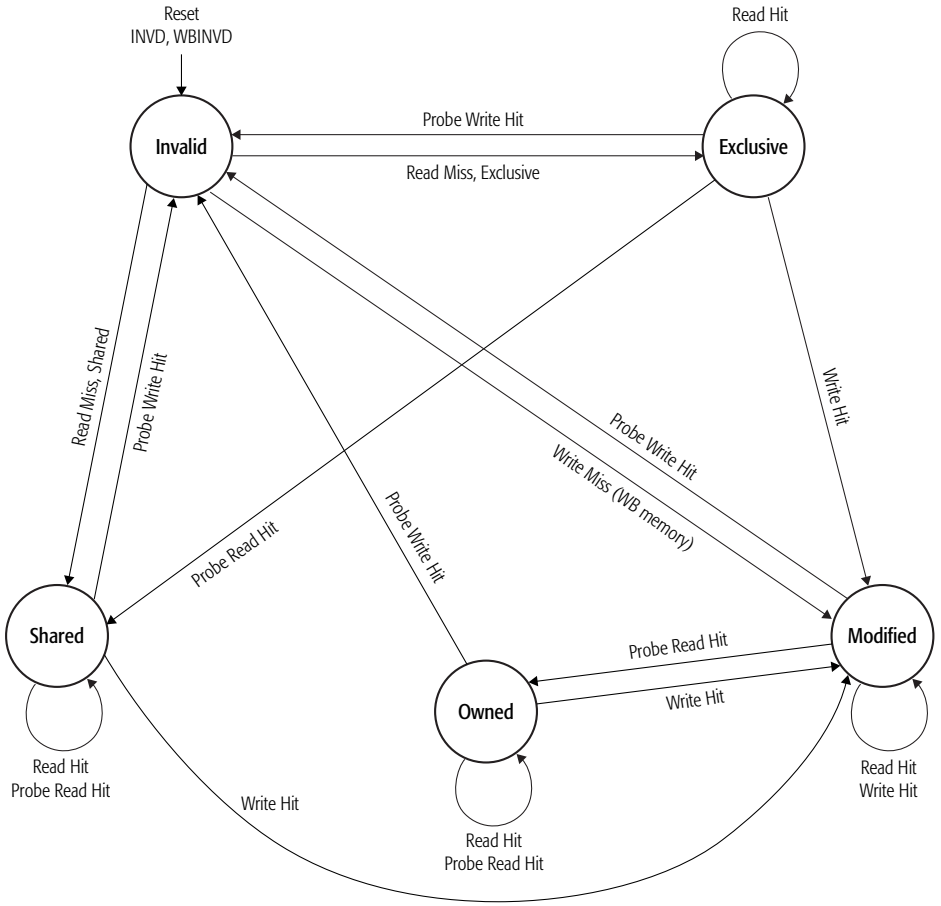
--

No action or signal produced

The MOESI Protocol

The Five state protocol based on MESI

Implemented in AMD64 line of multi core personal computers and servers.



Picture Courtesy of "*AMD64 Architecture Programmer's Manual Volume 2: System Programming*."AMD64 Technology. September 2006

MOESI States

Modify
- This line has the only valid copy
- Memory is stale

Owned
- This line has the valid copy. Other caches may have a “shared” copy of the data
- Memory is stale

Exclusive
- This line has a valid copy and no other cache have one
- Memory is up to date

Shared
- Data is replicated across many caches and memory
- Note: They may be many shared copies but only one (or zero) owned

Invalid

The Extra States Rationale

Exclusive (MSI to MESI)

- Reduce the number of busses transactions when a value is read exclusively and it may be modified in the future

Owned (MESI to MOESI)

- Reduce the number of busses transactions by delaying the update to the memory

Bibliography

Mosberger, David. "*Memory Consistency Models.*" Department of Computer Science. University of Arizona. November 1993.

Adve, Sarita; Gharachorloo, Kourosh. *"Shared Memory Consistency Models: A Tutorial."* Rice University and DEC. IEEE Transactiona on Computers. 1996

Lamport, Leslie. "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs." IEEE Transactions on Computers, September 1979, pp.690-691

"*AMD64 Architecture Programmer's Manual Volume 2: System Programming*."AMD64 Technology. September 2006

Lenoski, et. Al. "The Stanford DASH Multiprocessor." IEEE Computrer, 25(3): 63 – 69, March 1992